OIPE JC86
AUG 06 1999
PATENT & TRADEMARK OFFICE

APPENDIX A

```
#ifndef __AttrDef_hpp__
#define __AttrDef_hpp__

#include <iostream.h>

////////////////////////////////////////////////////////////
//                        AttrDef                         //
////////////////////////////////////////////////////////////

/*
 * Description:
 *   Attribute (member) definition. Contains RTTI and offset within the object
 *
 */

class RTTI;

class AttrDef
{
public:     //// Create

    AttrDef (const RTTI* rtti, int offset);

    virtual
    ~AttrDef ();

public:     //// Use
    int
    get_offset() const;

    const RTTI*
    get_rtti() const;

public:     //// Printing

    friend ostream&
    operator<< ( ostream& os, const AttrDef& obj );
    // Provide brief description of what is printed.

protected:

    virtual void
    print_on ( ostream& os ) const;
    // Called by the output operator.

private:    //// Attributes
    const RTTI* rtti_;
    int offset_;

}; // class AttrDef


#endif // __AttrDef_hpp__

```

```
 1: #include <AttrDef.hpp>
 2: #include <RTTI.hpp>
 3:
 4: ////////////////////////////////////////////////////////////
 5: //                        AttrDef                         //
 6: ////////////////////////////////////////////////////////////
 7:
 8: AttrDef::AttrDef (const RTTI* rtti, int offset) :
 9:   rtti_(rtti), offset_(offset)
10: {}
11:
12: AttrDef::~AttrDef ()
13: {}
14:
15: int
16: AttrDef::get_offset() const
17: {
18:     return offset_;
19: }
20:
21:
22: const RTTI*
23: AttrDef::get_rtti() const
24: {
25:     return rtti_;
26: }
27:
28:
29: void
30: AttrDef::print_on ( ostream& os ) const
31: {
32:     os << *rtti_ << " \t" << offset_;
33: }
34:
35: ostream&
36: operator<< ( ostream& os, const AttrDef& obj )
37: {
38:     obj.print_on ( os );
39:     return os;
40: }
```

```
 1: #ifndef __AttributeMap_hpp__
 2: #define __AttributeMap_hpp__
 3:
 4: #include <vector>
 5: #include <iostream.h>
 6: #include <AttrDef.hpp>
 7:
 8: //////////////////////////////////////////////////////////
 9: //                        AttributeMap
   //
10: //////////////////////////////////////////////////////////
11:
12: /*
13:  * Description:
14:  *   Abstrust class for Concrete Class Attribute Map (Pointer to Memeber Map).
15:  *  Concrete class' AttributeMaps are Singletons
16:  *
17:  */
18:
19: class AttributeMap
20: {
21: public:     //// Create
22:     virtual
23:     ~AttributeMap ();
24:
25: protected:     //// Create
26:     AttributeMap ();
27:
28: public:     //// Use
29:     const AttrDef&
30:     getAttrDef (int i) const;
31:
32:     int
33:     size () const;
34:
35: public:     //// Printing
36:     friend  ostream&
37:     operator<< ( ostream& os, const AttributeMap& obj );
38:     // Provide brief description of what is printed.
39:
40: protected:
41:
42:     virtual void
43:     print_on ( ostream& os ) const;
44:     // Called by the output operator.
45:
46: protected:    //// Attributes
47:     vector <AttrDef, allocator<AttrDef> > attr_map_;
48:
49: }; // class AttributeMap
50:
51: #endif // __AttributeMap_hpp__
52:
```

```
 1: #include <AttributeMap.hpp>
 2: #include <RTTI.hpp>
 3:
 4: ////////////////////////////////////////////////////////////
 5: //                        AttributeMap                    //
 6: ////////////////////////////////////////////////////////////
 7:
 8: AttributeMap::AttributeMap ()
 9: {}
10:
11: AttributeMap::~AttributeMap ()
12: {}
13:
14: const AttrDef&
15: AttributeMap::getAttrDef (int i) const
16: {
17:     return attr_map_[i];
18: }
19:
20: int
21: AttributeMap::size () const
22: {
23:     return attr_map_.size();
24: }
25:
26: void
27: AttributeMap::print_on ( ostream& os ) const
28: {
29:     if (attr_map_.size()) {
30:     os << "Type\tSize\tOffset" << endl;
31:
32:     for (int i = 0; i < attr_map_.size(); i++)
33:         os << attr_map_[i] << endl;
34:     }
35: }
36:
37: ostream&
38: operator<< ( ostream& os, const AttributeMap& obj )
39: {
40:     obj.print_on ( os );
41:     return os;
42: }
```

```
#ifndef __Externalizer_hpp__
#define __Externalizer_hpp__

#include <iostream.h>

////////////////////////////////////////////////////////////
//                      Externalizer                      //
////////////////////////////////////////////////////////////

/*
 * Description:
 *   Class to store any object in plain ASCII file
 *
 */

class RTTI;

class Externalizer
{
public:     //// Create
    Externalizer (const RTTI* rtti, ostream& os = cout);

    ~Externalizer ();

public:     //// Use
   void
   execute(void* ptr);

public:     //// Printing
    friend ostream&
    operator<< ( ostream& os, const Externalizer& obj );
    // Provide brief description of what is printed.

protected:
    virtual void
    print_on ( ostream& os ) const;
    // Called by the output operator.

private:    //// Attributes
    const RTTI* rtti_;

    ostream& os_;

private:    //// Implementation
    void
    byte_sequence(char* ptr, int size);

}; // class Externalizer

#endif // __Externalizer_hpp__
```

```
  1: #include <Externalizer.hpp>
  2: #include <AttributeMap.hpp>
  3:
  4: #include <RTTI.hpp>
  5:
  6: ////////////////////////////////////////////////////////////
  7: //                      Externalizer                      //
  8: ////////////////////////////////////////////////////////////
  9:
 10: Externalizer::Externalizer (const RTTI* rtti, ostream& os) :
 11:   rtti_(rtti), os_ (os)
 12: {}
 13:
 14: Externalizer::~Externalizer ()
 15: {}
 16:
 17: void
 18: Externalizer::execute ( void* ptr )
 19: {
 20:     int members = 0;
 21:
 22:     if (rtti_ -> isBuiltIn()) {
 23:         byte_sequence((char*)ptr, rtti_ -> getSize());
 24:         os_ << endl;
 25:         return;
 26:     }
 27:
 28:     for (int i = 0; i < rtti_ -> attribute_map() -> size(); i++) {
 29:         AttrDef def = rtti_ -> attribute_map() -> getAttrDef (i);
 30:         const RTTI* r = def.get_rtti ();
 31:         void* p = (char*)ptr + def.get_offset();
 32:
 33:         cout << i << ":-> "; //This output is for illustration
 34:
 35:         if (r -> isBuiltIn()) {
 36:             byte_sequence((char*)p, r -> getSize());
 37:             cout << endl; //This output is for illustration
 38:             os_ << endl;
 39:         } else {
 40:             Externalizer ex (r, os_);
 41:             ex.execute(p);
 42:             return;
 43:         }
 44:     }
 45: }
 46:
 47: void
 48: Externalizer::byte_sequence(char* ptr, int size)
 49: {
 50:     for (int j = 0; j < size; j++) {
 51:         os_ << (unsigned int)(unsigned char)*ptr++ << " ";
 52:         cout <<  (unsigned int)(unsigned char)*(ptr - 1) << " "; //This output
is for illustration
 53:     }
 54:
 55: }
 56:
 57: void
 58: Externalizer::print_on ( ostream& os ) const
 59: {
 60:     os << "Externalizer for:\nType\tSize" << endl
 61:       << *rtti_;
 62: }
 63:
 64: ostream&
 65: operator<< ( ostream& os, const Externalizer& obj )
 66: {
 67:     obj.print_on ( os );
 68:     return os;
 69: }
 70:
```

```
#ifndef __Internalizer_hpp__
#define __Internalizer_hpp__

#include <iostream.h>

class RTTI;

//////////////////////////////////////////////////////////////
//                        Internalizer                      //
//////////////////////////////////////////////////////////////

/*
 * Description:
 *    Restores an object from an ASCII stream
 *
 */

class Internalizer {
public:     //// Create
    Internalizer (const RTTI* rtti, istream& is = cin);
    virtual ~Internalizer();

public:     //// Use
    void
    execute(void* ptr);

private:    //// Attributes
    const RTTI* rtti_;
    istream& is_;

private:    //// Implementation
    void
    copy_bytes(char* ptr, int size);
};

#endif //__Internalizer_hpp__
```

```
#include <Internalizer.hpp>
#include <AttributeMap.hpp>
#include <RTTI.hpp>

////////////////////////////////////////////////////////////
//                      Internalizer                      //
////////////////////////////////////////////////////////////

Internalizer::Internalizer (const RTTI* rtti, istream& is) : rtti_(rtti), is_(is)
{}

Internalizer::~Internalizer()
{}

void
Internalizer::execute(void* ptr)
{
    int members = 0;

    if (rtti_ -> isBuiltIn()) {
        copy_bytes((char*)ptr, rtti_ -> getSize());
        return;
    }

    for (int i=0; i < rtti_ -> attribute_map() -> size(); i++) {
        AttrDef def = rtti_ -> attribute_map() -> getAttrDef(i);
        const RTTI* r = def.get_rtti();
        void* p = (char*)ptr + def.get_offset();

        if (r -> isBuiltIn()) {
            copy_bytes((char*)p, r -> getSize());
        } else {
            Internalizer in (r, is_);
            in.execute(p);
            return;
        }
    }
}

void
Internalizer::copy_bytes(char* ptr, int size)
{
    unsigned int tmp;
    for (int j = 0; j < size; j++) {
        is_ >> tmp;
        *ptr++ = char(tmp);
    }
}
```

```
 1: #ifndef __Object1_hpp__
 2: #define __Object1_hpp__
 3:
 4: #include <iostream.h>
 5: #include <Object2.hpp>
 6:
 7: //////////////////////////////////////////////////////////
 8: //                        Object1                       //
 9: //////////////////////////////////////////////////////////
10:
11: /*
12:  * Description:
13:  *   Test class for Standard Externalization Protocol Development
14:  *
15:  */
16:
17: class AttributeMap;
18:
19: class Object1
20: {
21: public:     //// Create
22:
23:     Object1 ();
24:
25:     Object1 (int n, float f, const Object2& e);
26:
27:     //    virtual
28:     ~Object1 ();
29:     //    ####################################
30:     //    ###   The following lines are generated by IDL compiler ###
31:     //    ####################################
32:     friend class Object1AttributeMap;
33:
34:     const static AttributeMap*
35:     attribute_map ();
36:     //    ####################################
37:     //    ###   The above lines are generated by IDL compiler ###
38:     //    ####################################
39:
40: public:     //// Printing
41:     friend ostream&
42:     operator<< ( ostream& os, const Object1& obj );
43:     // Provide brief description of what is printed.
44:
45: protected:
46:     virtual void
47:     print_on ( ostream& os ) const;
48:     // Called by the output operator.
49:
50: private:    //// Attributes
51:     int a_;
52:     float f_;
53:
54:     Object2 embedded_;
55:
56: }; // class Object1
57:
58: #endif // __Object1_hpp__
59:
```

```
 1: #include <Object1.hpp>
 2: #include <Object1AttributeMap.hpp>
 3:
 4: ////////////////////////////////////////////////////////////
 5: //                        Object1                         //
 6: ////////////////////////////////////////////////////////////
 7:
 8: //const AttributeMap* Object1::attr_map_ptr_ = 0;
 9:
10: Object1::Object1 () : a_(0), f_(0), embedded_(0)
11: {}
12:
13: Object1::Object1 (int n, float f, const Object2& e) :
14:   a_(n), f_(f), embedded_(e)
15: {}
16:
17: Object1::~Object1 ()
18: {}
19:
20: //    ####################################
21: //    ###   The following lines are generated by IDL compiler ###
22: //    ####################################
23: const AttributeMap*
24: Object1::attribute_map ()
25: {
26:     return Object1AttributeMap::map_instance();
27: }
28: //    ###################################
29: //    ###   The above lines are generated by IDL compiler ###
30: //    ###################################
31:
32: void
33: Object1::print_on ( ostream& os ) const
34: {
35:     os << "0:  " << a_ << "; "
36:        << "1:  " << f_ << "; "
37:        << "2:  " << embedded_;
38: }
39:
40: ostream&
41: operator<< ( ostream& os, const Object1& obj )
42: {
43:     obj.print_on ( os );
44:     return os;
45: }
```

```
 1: #ifndef __Object1AttributeMap_hpp__
 2: #define __Object1AttributeMap_hpp__
 3:
 4: #include <iostream.h>
 5:
 6: #include <AttributeMap.hpp>
 7:
 8: ////////////////////////////////////////////////////////////
 9: //                    Object1AttributeMap                 //
10: ////////////////////////////////////////////////////////////
11:
12: /*
13:  * Description:
14:  *   Map creation and management for Object1 class.
15:  *    ##############################
16:  *   ###   This class is generated by IDL compiler ###
17:  *   ##############################
18:  */
19:
20: class Object1;
21:
22: class Object1AttributeMap : public AttributeMap
23: {
24: private:     //// Create
25:     Object1AttributeMap ();
26:
27: public:     //// Create
28:     virtual
29:     ~Object1AttributeMap ();
30:
31:     static const AttributeMap*
32:     map_instance();
33:
34: public:     //// Printing
35:     friend ostream&
36:     operator<< ( ostream& os, const Object1AttributeMap& obj );
37:     // Provide brief description of what is printed.
38:
39: protected:
40:     virtual void
41:     print_on ( ostream& os ) const;
42:     // Called by the output operator.
43:
44: private:
45:     static AttributeMap* instance_;
46:
47: }; // class Object1AttributeMap
48:
49:
50: #endif // __Object1AttributeMap_hpp__
51:
```

```
   1: #include <Object1AttributeMap.hpp>
   2: #include <AttributeMap.hpp>
   3: #include <Object1.hpp>
   4: #include <Object2.hpp>
   5: #include <RTTIClassTemplate.hpp>
   6: #include <RTTIint.hpp>
   7: #include <RTTIfloat.hpp>
   8:
   9: ////////////////////////////////////////////////////////////
  10: //                      Object1AttributeMap                 //
  11: ////////////////////////////////////////////////////////////
  12:
  13: AttributeMap* Object1AttributeMap::instance_ = 0;
  14:
  15: Object1AttributeMap::Object1AttributeMap ()
  16: {
  17:     int offset = 0;
  18:     int const size = sizeof(Object1);
  19:     char tmp[size];
  20:
  21:     Object1* obj = (Object1*) tmp;
  22:
  23:     offset = (int)(&(obj->a_)) - (int)(obj);
  24:     attr_map_.push_back(AttrDef(new RTTIint (), offset));// Insert attribute (
member ) definition  (a_ is int).
  25:
  26:     offset = (int)(&(obj->f_)) - (int)obj ;
  27:     attr_map_.push_back(AttrDef(new RTTIfloat (), offset));// Insert attribute
( member ) definition (f_ is float).
  28:
  29:     offset = (int)(&(obj->embedded_)) - (int)obj;
  30:     attr_map_.push_back(AttrDef(new RTTIClassTemplate<Object2> ("Object2"), of
set));// Insert  attribute ( member ) definition  (embedded_ is Object2).
  31: }
  32:
  33: Object1AttributeMap::~Object1AttributeMap ()
  34: {}
  35:
  36: const AttributeMap*
  37: Object1AttributeMap::map_instance()
  38: {
  39:     if (instance_ == 0) {
  40:         instance_ = new Object1AttributeMap ();
  41:     }
  42:     return instance_;
  43: }
  44:
  45: void
  46: Object1AttributeMap::print_on ( ostream& os ) const
  47: {
  48:     AttributeMap::print_on(os);
  49: }
  50:
  51: ostream&
  52: operator<< ( ostream& os, const Object1AttributeMap& obj )
  53: {
  54:     obj.print_on ( os );
  55:     return os;
  56: }
```

```
#ifndef __Object2_hpp__
#define __Object2_hpp__

#include <iostream.h>

//////////////////////////////////////////////////////////////
//                          Object2                         //
//////////////////////////////////////////////////////////////

/*
 * Description:
 *   Test Object2 type
 *
 */

class AttributeMap;

class Object2
{
public:     //// Create

    Object2 ();

    Object2 (int n);

    virtual
    ~Object2 ();
    //    ####################################
    //    ###   The following lines are generated by IDL compiler ###
    //    ####################################
    friend class Object2AttributeMap;

    const static AttributeMap*
    attribute_map ();
    //    ####################################
    //    ###   The above lines are generated by IDL compiler ###
    //    ####################################

public:    //// Use
    void
    set (int n);

    int
    get ();

public:     //// Printing
    friend ostream&
    operator<< ( ostream& os, const Object2& obj );
    // Provide brief description of what is printed.

protected:
    virtual void
    print_on ( ostream& os ) const;
    // Called by the output operator.

private:    //// Attributes
    int n_;

}; // class Object2

#endif // __Object2_hpp__
```

```
 1: #include <Object2.hpp>
 2: #include <Object2AttributeMap.hpp>
 3:
 4: ////////////////////////////////////////////////////////////
 5: //                          Object2                       //
 6: ////////////////////////////////////////////////////////////
 7:
 8: Object2::Object2 () : n_(0)
 9: {}
10:
11: Object2::Object2 (int n) : n_(n)
12: {}
13:
14: Object2::~Object2 ()
15: {}
16:
17: //    ####################################
18: //    ###   The following lines are generated by IDL compiler ###
19: //    ####################################
20: const AttributeMap*
21: Object2::attribute_map ()
22: {
23:     return Object2AttributeMap::map_instance();
24: }
25: //    ####################################
26: //    ###   The above lines are generated by IDL compiler ###
27: //    ####################################
28:
29: void
30: Object2::set (int n)
31: {
32:     n_ = n;
33: }
34:
35: int
36: Object2::get ()
37: {
38:     return n_;
39: }
40:
41: void
42: Object2::print_on ( ostream& os ) const
43: {
44:     os << "O:  " << n_;
45: }
46:
47: ostream&
48: operator<< ( ostream& os, const Object2& obj )
49: {
50:     obj.print_on ( os );
51:     return os;
52: }
```

```
#ifndef __Object2AttributeMap_hpp__
#define __Object2AttributeMap_hpp__

#include <iostream.h>

#include <AttributeMap.hpp>

//////////////////////////////////////////////////////////////
//                          Object2AttributeMap
//
//////////////////////////////////////////////////////////////

/*
 * Description:
 *   Map creation and management for Object2 class.
 *    ###############################
 *   ###   This class is generated by IDL compiler ###
 *   ###############################
 *
 */

class Object2;

class Object2AttributeMap : public AttributeMap
{
private:   //// Create
    Object2AttributeMap ();

public:     //// Create
    virtual
    ~Object2AttributeMap ();

    static const AttributeMap*
    map_instance();

public:     //// Printing

    friend ostream&
    operator<< ( ostream& os, const Object2AttributeMap& obj );
    // Provide brief description of what is printed.

protected:

    virtual void
    print_on ( ostream& os ) const;
    // Called by the output operator.

private:
    static AttributeMap* instance_;

}; // class Object2AttributeMap


#endif // __Object2AttributeMap_hpp__
```

```
   1: #include <Object2AttributeMap.hpp>
   2: #include <AttributeMap.hpp>
   3: #include <Object2.hpp>
   4: #include <RTTIClassTemplate.hpp>
   5: #include <RTTIint.hpp>
   6:
   7: ////////////////////////////////////////////////////////////
   8: //                          Object2AttributeMap                          //
   9: ////////////////////////////////////////////////////////////
  10:
  11: AttributeMap* Object2AttributeMap::instance_ = 0;
  12:
  13: Object2AttributeMap::Object2AttributeMap ()
  14: {
  15:     int offset = 0;
  16:     int const size = sizeof(Object2);
  17:     char tmp[size];
  18:
  19:     Object2* obj = (Object2*) tmp;
  20:
  21:     offset = (int)(&(obj->n_)) - (int)obj ;
  22:     attr_map_.push_back(AttrDef(new RTTIint (), offset));// Insert attribute (
member ) definition  (a_ is int).
  23: }
  24:
  25: Object2AttributeMap::~Object2AttributeMap ()
  26: {}
  27:
  28: const AttributeMap*
  29: Object2AttributeMap::map_instance()
  30: {
  31:     if (instance_ == 0) {
  32:         instance_ = new Object2AttributeMap ();
  33:     }
  34:     return instance_;
  35: }
  36:
  37: void
  38: Object2AttributeMap::print_on ( ostream& os ) const
  39: {
  40:     AttributeMap::print_on(os);
  41: }
  42:
  43: ostream&
  44: operator<< ( ostream& os, const Object2AttributeMap& obj )
  45: {
  46:     obj.print_on ( os );
  47:     return os;
  48: }
```

```
#ifndef __RTTI_hpp__
#define __RTTI_hpp__

#include <iostream.h>

//////////////////////////////////////////////////////////////
//                          RTTI                            //
//////////////////////////////////////////////////////////////

class AttributeMap;

/*
 * Description:
 *   Abstruct class for RTTI
 *
 */

#include <string>

class RTTI
{
public:     //// Create
    RTTI (const char* t, bool builtIn);

    virtual
    ~RTTI ();

public:     //// Configuration
    virtual int
    getSize() const = 0;

    virtual const AttributeMap*
    attribute_map () const = 0;

    bool
    isBuiltIn() const;

    const string&
    getType() const;

public:     //// Printing
    friend ostream&
    operator<< ( ostream& os, const RTTI& obj );
    // Provide brief description of what is printed.

protected:
    virtual void
    print_on ( ostream& os ) const;
    // Called by the output operator.

private:    //// Attributes
    bool builtIn_;
    // It's 'true' for built-in types (int, float, double, char) and 'false' f
r user defined (classes, structures, unions)

    string type_;

}; // class RTTI

#endif // __RTTI_hpp__
```

```
#include <RTTI.hpp>

////////////////////////////////////////////////////////////
//                          RTTI                          //
////////////////////////////////////////////////////////////

RTTI::RTTI (const char* t, bool builtIn) : type_(string(t)), builtIn_(builtIn)
{}

RTTI::~RTTI ()
{}

const string&
RTTI::getType() const
{
    return type_;
}

bool
RTTI::isBuiltIn() const
{
    return builtIn_;
}

void
RTTI::print_on ( ostream& os ) const
{
    os << type_ << "\t" << getSize();
}

ostream&
operator<< ( ostream& os, const RTTI& obj )
{
    obj.print_on ( os );
    return os;
}
```

```
1: #ifndef __RTTIClassTemplate_hpp__
2: #define __RTTIClassTemplate_hpp__
3:
4:
5: #include <iostream.h>
6:
7: #include <RTTI.hpp>
8:
9: ////////////////////////////////////////////////////////////
10: //                        RTTIClassTemplate<T>                    //
11: ////////////////////////////////////////////////////////////
12:
13: /*
14:  * Description:
15:  *   Class template to create concrete RTTI  class objects
16:  *
17:  */
18:
19: class AttributeMap;
20:
21: template <class T>
22: class RTTIClassTemplate : public RTTI
23: {
24: public:     //// Create
25:     RTTIClassTemplate (const char* t);
26:
27:     virtual
28:     ~RTTIClassTemplate ();
29:
30: public:     //// Use
31:     virtual int
32:     getSize() const;
33:
34:     virtual const AttributeMap*
35:     attribute_map () const;
36:
37: public:     //// Printing
38:     friend  ostream&
39:     operator<< ( ostream& os, const RTTIClassTemplate<T>& obj );
40:     // Provide brief description of what is printed.
41:
42: protected:
43:     virtual void
44:     print_on ( ostream& os ) const;
45:     // Called by the output operator.
46:
47: }; // class RTTIClassTemplate<T>
48:
49: #ifndef NO_COMPILE_INSTANTIATE
50: #include <RTTIClassTemplate.cc>
51: #endif
52:
53: #endif // __RTTIClassTemplate_hpp__
54:
```

```
 1: #include <RTTIClassTemplate.hpp>
 2: #include <AttributeMap.hpp>
 3:
 4: ////////////////////////////////////////////////////////////
 5: //                        RTTIClassTemplate<T>                        //
 6: ////////////////////////////////////////////////////////////
 7:
 8: template <class T>
 9: RTTIClassTemplate<T>::RTTIClassTemplate (const char* t) :
10:   RTTI(t, false)
11: {}
12:
13: template <class T>
14: RTTIClassTemplate<T>::~RTTIClassTemplate ()
15: {}
16:
17: template <class T>
18: int
19: RTTIClassTemplate<T>::getSize() const
20: {
21:     return (sizeof (T));
22: }
23:
24: template <class T>
25: const AttributeMap*
26: RTTIClassTemplate<T>::attribute_map () const
27: {
28:     return T::attribute_map();
29: }
30:
31: template <class T>
32: void
33: RTTIClassTemplate<T>::print_on ( ostream& os ) const
34: {
35:     RTTI::print_on(os);
36: }
37:
38: template <class T>
39: ostream&
40: operator<< ( ostream& os, const RTTIClassTemplate<T>& obj )
41: {
42:     obj.print_on ( os );
43:     return os;
44: }
```

```
#ifndef __RTTIfloat_hpp__
#define __RTTIfloat_hpp__

#include <iostream.h>

#include <RTTI.hpp>

//////////////////////////////////////////////////////////////
//                          RTTIfloat                       //
//////////////////////////////////////////////////////////////

/*
 * Description:
 *   RTTI for boilt-in 'float' type
 *
 */

class RTTIfloat : public RTTI
{
public:     //// Create

    RTTIfloat ();

    virtual
    ~RTTIfloat ();


public:     //// Use
    virtual int
    getSize() const;

    virtual const AttributeMap*
    attribute_map () const;

public:     //// Printing
    friend ostream&
    operator<< ( ostream& os, const RTTIfloat& obj );
    // Provide brief description of what is printed.

protected:
    virtual void
    print_on ( ostream& os ) const;
    // Called by the output operator.

}; // class RTTIfloat


#endif // __RTTIfloat_hpp__
```

```
 1: #include <RTTIfloat.hpp>
 2:
 3: ////////////////////////////////////////////////////////
 4: //                        RTTIfloat                   //
 5: ////////////////////////////////////////////////////////
 6:
 7:
 8: RTTIfloat::RTTIfloat () : RTTI("float", true)
 9: {}
10:
11: RTTIfloat::~RTTIfloat ()
12: {}
13:
14: int
15: RTTIfloat::getSize() const
16: {
17:     return (sizeof (float));
18: }
19:
20: const AttributeMap*
21: RTTIfloat::attribute_map () const
22: {
23:     throw ("Built-in 'float' type has no attribute map");
24: }
25:
26: void
27: RTTIfloat::print_on ( ostream& os ) const
28: {
29:     RTTI::print_on(os);
30: }
31:
32: ostream&
33: operator<< ( ostream& os, const RTTIfloat& obj )
34: {
35:     obj.print_on ( os );
36:     return os;
37: }
```

```
 1: #ifndef __RTTIint_hpp__
 2: #define __RTTIint_hpp__
 3:
 4: #include <iostream.h>
 5:
 6: #include <RTTI.hpp>
 7:
 8: ////////////////////////////////////////////////////////////
 9: //                        RTTIint                         //
10: ////////////////////////////////////////////////////////////
11:
12: /*
13:  * Description:
14:  *   RTTI for boilt-in 'int' type
15:  *
16:  */
17:
18: class RTTIint : public RTTI
19: {
20: public:     //// Create
21:
22:     RTTIint ();
23:
24:     virtual
25:     ~RTTIint ();
26:
27:
28: public:     //// Use
29:     virtual int
30:     getSize() const;
31:
32:     virtual const AttributeMap*
33:     attribute_map () const;
34:
35: public:     //// Printing
36:     friend ostream&
37:     operator<< ( ostream& os, const RTTIint& obj );
38:     // Provide brief description of what is printed.
39:
40: protected:
41:     virtual void
42:     print_on ( ostream& os ) const;
43:     // Called by the output operator.
44:
45: }; // class RTTIint
46:
47:
48: #endif // __RTTIint_hpp__
49:
```

```
 1: #include <RTTIint.hpp>
 2:
 3: ////////////////////////////////////////////////////////
 4: //                        RTTIint                     //
 5: ////////////////////////////////////////////////////////
 6:
 7:
 8: RTTIint::RTTIint () : RTTI("int", true)
 9: {}
10:
11: RTTIint::~RTTIint ()
12: {}
13:
14: int
15: RTTIint::getSize() const
16: {
17:     return (sizeof (int));
18: }
19:
20: const AttributeMap*
21: RTTIint::attribute_map () const
22: {
23:     throw ("Built-in 'int' type has no attribute map");
24: }
25:
26: void
27: RTTIint::print_on ( ostream& os ) const
28: {
29:     RTTI::print_on(os);
30: }
31:
32: ostream&
33: operator<< ( ostream& os, const RTTIint& obj )
34: {
35:     obj.print_on ( os );
36:     return os;
37: }
38:
```

```
  1: #include <iostream.h>
  2: #include <strstream.h>
  3:
  4: #include <Object1.hpp>
  5: #include <Externalizer.hpp>
  6: #include <Internalizer.hpp>
  7: #include <RTTIClassTemplate.hpp>
  8: #include <RTTIfloat.hpp>
  9:
 10: int
 11: main ()
 12: {
 13:     cout << endl;
 14:     cout << "Creating an object of user defined Object1 type object" << endl;
 15:
 16:     Object1 obj(3, 7, 21);
 17:     const AttributeMap* map = Object1::attribute_map();
 18:
 19:     cout << "\nObject1: " << obj << endl;
 20:
 21:     cout << "\nAttribute Map of the Object1: " << endl;
 22:     cout << *map << endl;
 23:
 24:     cout << "Creating an Externalizer for a given type (Object1)" << endl;
 25:
 26:     char temp[1024];
 27:     ostrstream ostr(temp, sizeof(temp));
 28:
 29:     Externalizer ext_obj (new RTTIClassTemplate<Object1> ("Object1"), ostr);
 30:     cout << ext_obj << endl;
 31:
 32:     cout << "\nExternalizing raw memory that contains an object of given type
Object1 type object)" << endl;
 33:     ext_obj.execute((void*) &obj);
 34:
 35:     cout << "Done" << endl;
 36:
 37:     cout << "\nCreating an Internalizer for a given type (Object1)" << endl;
 38:
 39:     istrstream istr(temp, ostr.pcount());
 40:
 41:     Internalizer int_obj(new RTTIClassTemplate<Object1> ("Object1"), istr);
 42:     Object1 obj_result;
 43:
 44:     cout << "Internalizing from raw memory into Object1" << endl;
 45:     int_obj.execute((void*) &obj_result);
 46:     cout << "Result " << obj_result << endl;
 47:
 48:     cout << "\nCreating an object float f = 3" << endl;
 49:
 50:     float f = 3;
 51:
 52:     cout << "\nCreating an Externalizer for a given type (float)" << endl;
 53:     char temp2[1024];
 54:     ostrstream ostr2(temp2, sizeof(temp2));
 55:
 56:     Externalizer ext_f (new RTTIfloat, ostr2);
 57:     cout << ext_f << endl;
 58:
 59:     cout << "\nExternalizing raw memory that contains an object of given type
float type object)" << endl;
 60:     ext_f.execute((void*) &f);
 61:     cout << "\nDone" << endl;
 62:
 63:     cout << "\nCreating an Internalizer for a given type (float)" << endl;
 64:     istrstream istr2(temp2, ostr2.pcount());
 65:     Internalizer int_f(new RTTIfloat, istr2);
 66:     float flt;
 67:
 68:     cout << "Internalizing  from raw memory into float" << endl;
 69:     int_f.execute((void*) &flt);
 70:     cout << "Result " << flt << endl;
 71:
 72:     return 0;
 73: }
```

```
PRINTOUTS.TXT
Creating an object of user defined Object1 type object

Object1: 0:  3; 1:  7; 2:  0:  21

Attribute Map of the Object1:
Type  Size  Offset
int   4  0
float 4  4
Object2  8  8

Creating an Externalizer for a given type (Object1)
Externalizer for:
Type  Size
Object1  20

Externalizing raw memory that contains an object of given type (Object1 type
object)
0:-> 0 0 0 3
1:-> 64 224 0 0
2:-> 0:-> 0 0 0 21
Done

Creating an Internalizer for a given type (Object1)
Internalizing from raw memory into Object1
Result 0:  3; 1:  7; 2:  0:  21

Creating an object float f = 3

Creating an Externalizer for a given type (float)
Externalizer for:
Type  Size
float 4

Externalizing raw memory that contains an object of given type (float type
object)
64 64 0 0
Done

Creating an Internalizer for a given type (float)
Internalizing  from raw memory into float
Result 3
```